Preprint of the paper which appeard in:
Proc. 1st IFAC Workshop on Multivehicle Systems, Salvador, Brazil, pp. 51-56, October 2-3, 2006.

REUSABLE ARCHITECTURE AND TOOLS FOR TEAMS OF LIGHTWEIGHT HETEROGENEOUS ROBOTS

Martin Friedmann, Jutta Kiener, Sebastian Petters, Dirk Thomas, Oskar von Stryk

Simulation and Systems Optimization Group, Department of Computer Science, Technische Universität Darmstadt, Hochschulstr. 10, D-64289 Darmstadt, Germany
www.sim.tu-darmstadt.de

Abstract: The software framework *RoboFrame* has been designed to meet the special requirements for teams of lightweight autonomous heterogeneous robot systems. Due to platform abstraction and modern object oriented design, it allows the reuse of components of common robot control software. It can also efficiently be implemented on new platforms and enables different control architectures for different tasks. For the exemplary application in autonomous robot soccer teams configurable and portable algorithms for vision, world modeling, behavior and motion control have been developed on top of the framework. For debugging, controlling and monitoring, an extendable graphical user interface and a generic simulator package have been implemented around the framework. Based on these instruments, different applications for homogeneous and heterogeneous robot teams can be realized in short time.

Keywords: Reusable robotic software, lightweight mobile autonomous robots, cooperation of heterogeneous robots, tools for debugging and monitoring, robot simulation.

1. INTRODUCTION

During the last decade there was an increased interest in cooperative heterogeneous mobile robot systems. Also driven by increasingly more advanced and powerful sensors and actuators more sophisticated tasks like rescuing people in hazardous area, observing a burning field (Ollero *et al.*, 2005) can be addressed including highly dynamic scenarios as the annual robot soccer competition (*RoboCup*, n.d.). The underlying program code to accomplish such difficult tasks is very complex and efficient tools for designing and maintaining control software for different applications are needed.

In this paper we describe our approach to encounter the inherent complexity in developing robot control software for teams of lightweight autonomous robots. The approach is mainly targeted to small robotic systems in the low and mid price range which are characterized by only a small payload and the requirement of stabilizing locomotion dynamics using inertial sensors like gyroscopes and accelerometers as it is the case for humanoid robots and unmanned aerial vehicles and also for small unmanned marine and offroad vehicles to some extent. Small bipedal robots as well as unmanned aerial vehicles may carry only an onboard computer with relatively low computational power and energy consumption but must maintain stability during locomotion. The lightweight aspect also refers to software and components for sensory perception, which can be executed with a good performance on such a robot.

As basis for developing control software for different applications the framework *RoboFrame* programmed in ANSI C++ has been designed. The implementation makes use of modern object oriented design techniques, like generic programming, multiple inheritance and common design patterns. From a software technology point of view

this allows the implementation of very low coupled algorithms for various subtasks like vision, localization, behavior control and motion, which enables fast adaption and recomposition for different kinds of robots.

RoboFrame also contains an extendable graphical user interface which supports development by high-level visualization of all data. Also meta information such as process layout and execution time can be visualized and edited during runtime. For tests without the real hardware, a generic simulation package has been developed which allows hard- and software tests in the loop.

The combination of a flexible software framework in conjunction with tools for debugging and maintenance actively supports and accelerates development of robot control software.

2. STATE OF RESEARCH

Currently used tools in software development for mobile robots cover only partly the special requirements of heterogeneous lightweight autonomous robots. E.g. the widely used mobile platform Pioneer 2dx (ActivMedia, 2002) is equipped with software and a GUI for vision, navigation, localization, and mapping. However, the software is only partly open source. Therefore, it cannot be adapted to new sensor hardware, and only supports the specific wheeled mobile platform.

The multi robot middleware system MIRO (Utz *et al.*, 2004) is primary designed for robots with multiple sensors with own processing units. It has mechanisms for logging single and multiple channel information. Sequences of a robot soccer game can be recorded and replayed for later analysis with a log player. The base framework and the applied tools are optimized for a CPU with a high computational power, thus limiting their usability on lightweight systems with limited payload.

For the four legged robots AIBO a robot control tool box with many application has been developed by the GermanTeam (Röfer *et al.*, 2005), which is supported only for Windows. The added dynamics simulator (Laue *et al.*, 2005) can handle not only the four legged robots, but also different legged and wheeled robot types.

The advanced dynamics simulator tool for humanoid robots (Kuffner *et al.*, 2003) is not presented with a logging option for exchanged data on the real robot system thus not supporting the development process of robot control software for new applications, nor information on the framework is given.

The Coupled Layer Architecture for Robotic Autonomy (CLARAty) represents a framework for reusable robotic components (Nesnas *et al.*, 2006). It allows the simple integration of new technologies to existing robots. CLARAty's main focus are wheeled robots, i.e. for exploration of unknown territories.

The framework *Alliance* (Parker, 1998) is applied mainly on wheeled robot systems. The software is fault-tolerant designed and is modular to sensor extension. In the project *Centibots* (Konolige, 2002) the framework for a very large team of robots is developed. The software is designed to execute basic structured tasks on the robots as exploration of an unknown environment. Adaption to heterogenous lightweight robots is not easily possible. The main focus in the architecture for tightly coupled multi-robot cooperation by (Chaimowicz *et al.*, 2001) is on the role distribution for wheeled robots. The adaptability to additional sensors or a different locomotion system by legs or in the air is not described. For none of these three projects supporting tools for the software and control architecture are presented in the literature.

Framework	MIRO	GT	CLARAty	Player/Stage
lightweight platforms	–	o	–	-
monitoring capabilities	+	+	+	+
different OS	+	–	+	–
robot simulator	–	+	+	+
modularity, reuseability	+	+	+	+

Table 1. Comparison of existing frameworks with different features.

The open source project Player/Stage consists of two parts: a robot control interface running as a server on a Linux or Unix based robot and a simulation backend. Its main focus is the research in robot and sensor systems. The separation into two parts causes an additional computational overhead, which is not adequate for lightweight autonomous robots. These samples of frameworks and tools for mobile robots all have special features and advantages, however none of it comes with features of controlling and monitoring a software and control architecture with low computational power as found in lightweight systems, running on different operating systems. Different frameworks are compared in Tab. 1.

3. FRAMEWORK AND MODULES

To support and ease the development of robot control software among several different platforms the object oriented framework *RoboFrame* has been developed to be used on various lightweight platforms and operating systems (OS). This has been achieved by implementing a thin hardware and OS abstraction layer providing an unified interface for network and multithreading services below the framework's platform independent core. To meet the special requirements of controlling potentially instable walking or flying systems, execution under realtime constraints is guaranteed if provided by the underlying OS.

When porting the framework to a new platform, only the abstraction layer has to be adapted. Currently the framework has been ported to Linux (x86, MIPS), BSD Unix, Windows 2000/XP and the realtime OS Windows CE. It is used on three different types of small humanoid robots (with

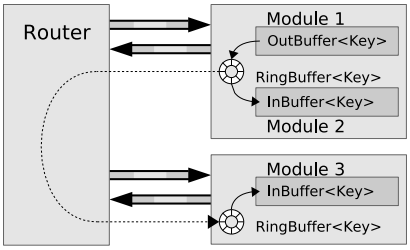

Fig. 1. Message flow within the framework.

Linux and Windows CE) and the wheeled robot Pioneer 2dx (with Windows) (Fig. 5). For testing with the robot simulator (cf. Sect. 4.2), it can be compiled and executed on Windows XP.

3.1 Architecture of Applications

The framework does not impose any special control paradigm on the developer. The software modules can be arranged not only in a hierarchical deliberative structure, but also in almost any other behavior control architecture. Execution of the modules and data exchange between them is controlled by the framework.

An application may consist of one or more threads of execution which may be distributed among several CPUs if available. Any number of modules may be executed sequentially within one thread.

Data exchange between the modules is provided by a messaging system (see Fig. 1). Each module declares which kind of messages it may receive and send and is provided with a buffer for each kind of messages. The framework transparently transfers any message emitted by one module to all modules receiving this kind of message. If the emitting and receiving module reside within the same thread, the messages need not be copied due to a special buffer allocation scheme which merges outgoing and ingoing buffers for the same kind of message within one thread. Messages are only transferred from one thread to another, if one of the modules of the receiving thread has a suitable incoming buffer.

Due to this flexible messaging system very low coupling between the modules is achieved and a robot control software can be extended easily by adding further modules. As there are no predefined communication paths, any newly added module can receive any kind of message existing in the application. Any module may be used in a new context as long as the input messages needed by this module are provided by some other module. This enables an easy integration of other kinds of software, e.g. graphical user interfaces (cf. Sect. 4.1) or simulations (cf. Sect. 4.2).

3.2 Modules

Using *RoboFrame* several platform independent modules have been developed which can be integrated into various robot control applications. Among others, there exist modules for the tasks of

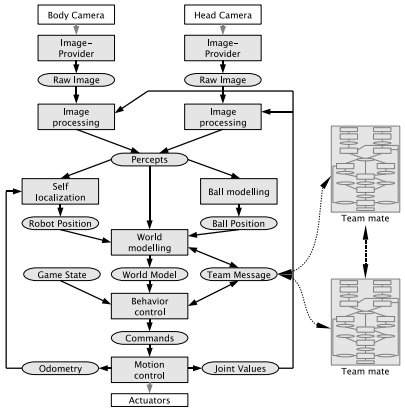

Fig. 2. Architecture of the control software used for the application *robot soccer* (rectangles describe modules, ovals describe messages).

vision, world modeling, robot behavior and motion control. Due to modern object oriented design, they are easily adjustable to work in different configurations and domains without the need to modify the source code while on the other hand allowing robust and fast execution. The modules have been integrated into the humanoid robot control architecture used at RoboCup 2006 (Fig. 2).

3.2.1. Vision The vision system handles camera data and provides the following modules with information about objects detected and their relative position to the robot. It consists of three modules for image acquisition, color segmentation and object recognition. Only image acquisition is hardware dependent and requires knowledge about the used camera and the image resolution. Currently, three different camera types are supported, others can be added easily without changing the higher levels of vision. The use of abstraction layers allows subsequent modules to be independent of these configurations, but if required, it is still possible to integrate domain specific knowledge to gain better detection results. Adding or removing components for the objects of interest allows a quick adaption of object recognition to different tasks.

3.2.2. World modeling World modeling uses data from image processing to calculate the pose of the robot and the position of obstacles. In a soccer scenario also position and speed of the ball is modeled and predicted with a Kalman filtering technique. For the task of self localization Markov localization with particle filtering (Fox *et al.*, 1999) is used. Any information generated by the world modeling module of one robot can be distributed to all collaborating robots enabling improved team cooperation.

3.2.3. Behavior Depending on the world model data, the behavior module generates requests for different actions of the robot. The robot's behavior is controlled by a hierarchy of state machines which is described using the language XABSL (Loetzsch *et al.*, 2006). Figure 3 shows a visualization of an option and the following actions.

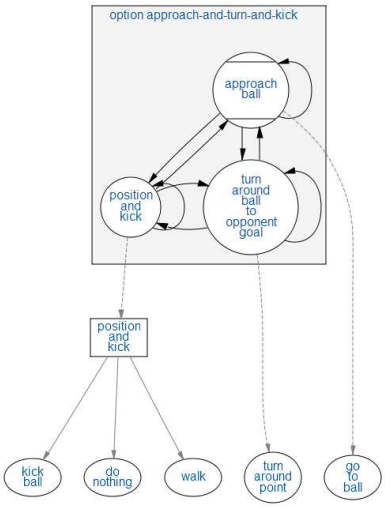

Fig. 3. A XABSL state machine (gray) describing the behavior for approaching the ball and positioning towards the opponent's goal. When needed, basic behaviors (oval) or other state machines (rectangles) are activated.

3.2.4. Motion The motion module directly communicates with the hardware of the robot. It receives motion requests from the behavior and transforms them into control messages for the robot. Further it reads the motor encoder and inertial sensors and transfers their data to the other modules. These tasks are highly dependent on the robot in use and thus the module has to be adapted for each new robot.

3.3 Communication

For communication between multiple robots, connection oriented TCP or datagram based UDP can be used. Depending on the requirements, the communication can be established on different abstraction levels and with different types of transparency. This way it is either possible to use data from a remote system as if it came from the own sensors or to differ explicitly between messages from different robots.

4. DEVELOPMENT TOOLS

Beside the application running on the robot, tools for monitoring the state of the robot, both concerning hardware and software, are required. These include tools for development, test and validation of functions for coordination and cooperation of autonomous robot teams as well as functions of individual robots. Efficient tools enable the developers to gain a higher insight into complex processes thus leading to better results in shorter time.

4.1 Graphical user interface

The framework provides a graphical user interface (GUI), which uses the same communication mechanisms as the framework described in the previous section. The GUI can connect with multiple robots simultaneously via TCP based data connections and exchange bidirectionally any kind of data. To ensure platform independency for the GUI, it is based on the C++ GUI toolkit Qt from Trolltech, which is available for various platforms.

The GUI is used for debugging, monitoring and controlling and can be extended with dialogs for any kind of visualization or interaction. Dialogs share the same principle as modules to be able to receive and send any kind of data. The API allows extending the graphical user interface with own dialogs. The framework includes some general-purpose dialogs for frequent scenarios.

The message recording dialog can be used to store any kind of data emitted by a running application to a binary log file.These log files can be investigated offline for debugging or be replayed to an application. This feature allows an easy comparison of different modules accomplishing the same task by providing them with the same input data and comparing their respective output data.

The chronometer dialog visualizes the runtime and call frequency of processes and modules of an attached application.Modules can even provide detailed zoning of their execution time. This feature allows detection of performance bottlenecks and therefore performance optimizations in the specific parts of the source code. Because of the amount of data accruing during profiling, these information are generated on demand only.

Another dialog is capable of visualizing the process layout of an attached application in a tree-based view. Furthermore the timings of the processes can be adjusted and modules can be disabled and re-enabled during runtime. This feature allows for faster testing of different process layouts and timings.

Outside the framework multiple dialogs have been created to allow debugging and visualizing the various algorithms used in the RoboCup specific modules.

4.2 Robot simulation

It has already been shown that it is easy to receive messages from and send messages to a running robot control application for controlling and debugging the application. The communication mechanism further on can be used to connect the application to a simulation software replacing some or all parts of the real robot hardware in various configurations.

Currently the simulation's main task is testing of the robot's behavior. In robotic soccer this requires simulation of the robot's cameras and motions (see Fig. 4). Due to the camera's motion, a 2D simulation as in the Player/Stage project (Gerkey *et al.*, 2003) does not suffice to generate the needed input-data, thus a 3D simulation has been developed. In contrast to other 3D simulation systems like Gazebo (Koenig and Howard, 2004), SimRobot (Laue *et al.*, 2005) or

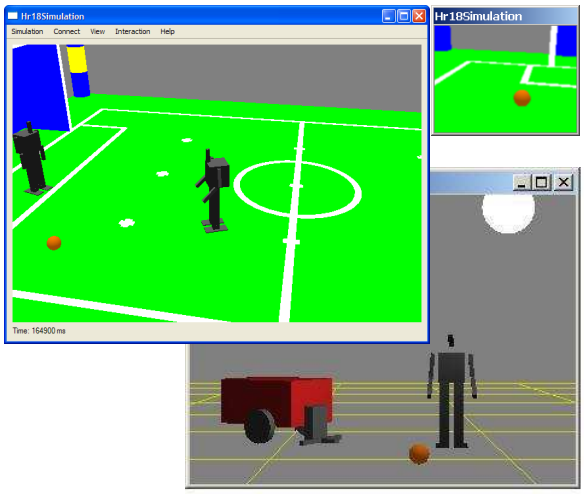

Fig. 4. Top: Simulated striker robot approaching the ball (left) and image of the simulated head camera (right). Bottom: Simulation of humanoid and a wheeled Pioneer 2dx with gripper performing a cooperative task.

UCHILSIM (Zagal and Ruiz-del Solar, 2004) the simulation is not based on the robot's dynamic. Instead the motion simulation is based on the robot's kinematics with the additional constraint, that at least one foot is touching the ground each moment. This leads to a less accurate model of the motion, but avoids the simulated robot from falling over, thus enabling extended tests of the higher level control software like behavior and vision.

On a system equipped with an Intel Centrino Duo (1.666 GHz) and standard chipset graphics (Intel 945GM Express), it is possible to simulate two teams of 4 humanoid robots with two cameras per robot in realtime using only one of the cpu's cores. The rate of the motion-simulation is 100 steps per second, each simulated camera generates 10 frames per second. The simulator is not limited to humanoid robots and soccer application. Also other applications and wheeled robots are possible (see Fig. 4).

5. APPLICATIONS

In the authors' group extensive use has been made of the described framework and supporting tools. Some applications are described in the following.

5.1 Benchmarking and parameter tuning of world modeling

The world modeling module only depends on motion information of the hardware of the robot and the output data of the vision module. This information is merged in a statistical model of the pose and environment of the robot.

To test various versions and parameter sets of the self localization, all motion and camera data of the physical robot are recorded while the robot is performing several actions (walking straight forward, turning in place, etc.). At the same time the robot pose is measured. Later the generated

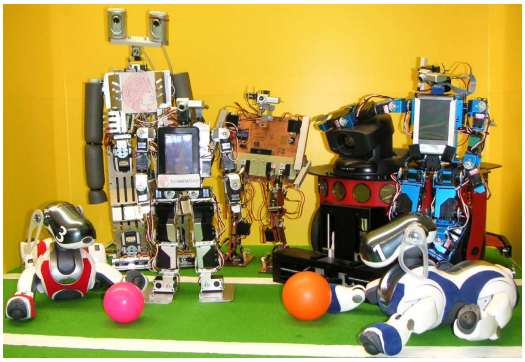

Fig. 5. Legged and wheeled robot systems, supported by the framework.

log files are replayed to the different version of the module and the modules output is compared to the real position of the robot measured during recording.

A similar approach is taken to optimize the parameters of Kalman filter used for ball modeling. Again all motions and camera images of the robot are recorded, while the ball is moved along a predefined trajectory on the playing field.

5.2 Developing robot and team behavior

The selection of robot motions is mainly determined by the behavior module. An extensive testing of this module is crucial for robot performance in solving a certain task.

Testing of behavior control in simulation. Due to the limited availability of real robots and the high strain on the robot hardware during extensive tests, it is desirable to have other means for testing the behavior control software. As the behavior directly reacts to the changing situation depending on the robot motion, approaches using log files as described in Sect. 5.1 are not feasible. This problem is overcome by the use of the robot simulation described in Sect. 4.2 in combination with the framework. Debugging the behavior on a real robot is very difficult, as undesired actions may be caused not only by undesired effects in the behavior control software but also by disturbances as slipping of the robot or distorted camera images. By using the simulation these additional sources of error are eliminated and debugging of the plain behavior control software is enabled. This approach also enables multiple developers simultaneously testing robot behavior regardless of the availability of real robots. Also different scenarios can be easily set up without the need for real objects leading to reduced hardware and material costs.

Soccer. Developing of a behavior for a team of heterogeneous robots requires knowledge of all robot states. With the mechanisms of RoboFrame and the graphical user interface, two dialogs have been designed for controlling and monitoring, which significantly support to developers. All data used to find a decision in the XABSL behavior can be visualized. It is also possible to modify the robot's decision tree to test specific situations.

Fig. 6. Two robots of the authors' team (left and right) during *RoboCup Japan Open* 2006 in 2-on-2 soccer game against Team Osaka.

Remote debugging and development. If several developers of robot control software and modules for the same application are working together spatially separated then communication between them is very important. Using log files with recorded data allows analysis of robot performance on distant places to the real robot and environment. During *RoboCup Japan Open* 2006 it was possible to optimize and adjust the robot control software and modules remotely thus reducing the number of developers needed on site.

Application to different robot types. The modularity of the framework and easy exchange of the robot platform could be demonstrated with different robots and operating systems: the wheeled robot platform Pioneer 2dx and Windows 2000 (achieved within a few days only) and small humanoid robots with Linux Mipsel or Windows CE (Fig. 5).

6. CONCLUSIONS AND OUTLOOK

An approach to handle the difficulties in developing robot control software and modules for teams of lightweight autonomous heterogeneous robots has been presented based on the framework *RoboFrame*. The reuse of algorithms for vision, world modeling, behavior and motion control on different platforms and the support of the developers in monitoring and debugging with the herein designed graphical instruments has been described. Several applications in robot soccer and beyond demonstrate the practical usability of the framework and the tools built on top of it.

During the RoboCup Japan Open 2006 the authors' team participated successfully in the 2-on-2 competitions (Fig. 6), during the RoboCup 2006 a 3-on-3 demonstration game was performed.

Midterm objectives are the extension and implementation of the framework and tools on lightweight outdoor robotic systems for the use in autonomous search and rescue operations. For this purpose, small unmanned marine and offroad ground vehicles are currently investigated.

REFERENCES

ActivMedia (2002). Pioneer 2, Mobile Robots, Operational Manual for Pioneer 2dx.

Chaimowicz, L., T. Sugar, V. Kumar and M. Campos (2001). An architecture for tightly coupled multi-robot cooperation. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. Vol. 3. pp. 2292–2297.

Fox, D., W. Burgard, F. Dellaert and S. Thrun (1999). Monte carlo localization: Efficient position estimation for mobile robots. In: *Proc. National Conf. on Artificial Intelligence*.

Gerkey, B. P., R. T. Vaughan and A. Howard (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In: *Intl. Conf. on Advanced Robotics (ICAR)*. Coimbra, Portugal. pp. 317 – 323.

Koenig, N. and A. Howard (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. Vol. 3. pp. 2149 – 2154.

Konolige, K. (2002). Saphira robot control architecture. Technical report. SRI International.

Kuffner, J., K. Nishiwaki, S. Kagami, M. Inaba and H. Inoue (2003). Motion planning for humanoid robots. In: *International Symposium of Robotics Research (ISRR)*.

Laue, T., K. Spiess and T. Röfer (2005). SimRobot: A general physical robot simulator and its application in RoboCup. In: *RoboCup 2005: Robot Soccer World Cup IX*.

Loetzsch, M., M. Risler and M. Jüngel (2006). XABSL – a pragmatic approach to behavior engineering. In: *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*. Beijing.

Nesnas, I. A., R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guinea, M.l McHenr, I. Shu and D. Apfelbaum (2006). Claraty: Challenges and steps toward reusable robotic software. *Intl. J. of Advanced Robotic Systems* **3**(1), 23 – 30.

Ollero, A., S. Lacroix, L. Merino, J. Gancet, J. Wiklund, V. Remuss, I. Veiga, L.G. Gutierrez, D.X. Viegas, M.A. Gonzalez, A. Mallet, R. Alami, G. Hommel R. Chatila, F.J. Colmenero, B. Arrue, J. Ferruz, R. Martinez de Dios and F. Caballero (2005). Architecture and perception issues in the comets multi-uav project. *IEEE Robotics and Automation Magazine* **12, issue 2**, 1–1.

Parker, L. E. (1998). ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation* **14**(2), 220–240.

RoboCup (n.d.). www.robocup.org.

Röfer, T. et al. (2005). German Team RoboCup 2005 - Team Report. Technical report. TU Darmstadt, Uni Bremen, HU Berlin, Uni Dortmund, see also www.germanteam.org.

Utz, H., G. Mayer and G. Kraetzschmar (2004). Middleware logging facilities for experimentation and evaluation in robotics. In: *27th German Conf. on Artificial Intelligence (KI)*.

Zagal, J. C. and J. Ruiz-del Solar (2004). UCHILSIM: A dynamically and visually realistic simulator for the robocup four legged league. In: *RoboCup 2004: Robot Soccer World Cup VIII*.